10-02-00

A

JC920 U.S. PTO 09/29/00

Please type a plus sign (+) inside this box [+]

PTO/SB/05 (12/97)
Approved for use through 09/30/00. OMB 0651-0032
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Attorney Docket No. 042390.P7512 Total Pages 2

First Named Inventor or Application Identifier Orna Etzion

Express Mail Label No. EL 617 211 053

JC836 U.S. PTO 09/676175 09/29/00

ADDRESS TO: **Assistant Commissioner for Patents**
Box Patent Application
Washington, D. C. 20231

APPLICATION ELEMENTS
See MPEP chapter 600 concerning utility patent application contents.

1. X Fee Transmittal Form
(Submit an original, and a duplicate for fee processing)

2. X Specification (Total Pages 18)
(preferred arrangement set forth below)
- Descriptive Title of the Invention
- Cross References to Related Applications
- Statement Regarding Fed sponsored R & D
- Reference to Microfiche Appendix
- Background of the Invention
- Brief Summary of the Invention
- Brief Description of the Drawings (if filed)
- Detailed Description
- Claims
- Abstract of the Disclosure

3. X Drawings(s) (35 USC 113) (Total Sheets 4)

4. ___ Oath or Declaration (Total Pages ___)
   a. ___ Newly Executed (Original or Copy)
   b. ___ Copy from a Prior Application (37 CFR 1.63(d))
   (for Continuation/Divisional with Box 17 completed) **(Note Box 5 below)**
   i. ___ DELETIONS OF INVENTOR(S) Signed statement attached deleting inventor(s) named in the prior application, see 37 CFR 1.63(d)(2) and 1.33(b).

5. ___ Incorporation By Reference (useable if Box 4b is checked)
The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.

6. ___ Microfiche Computer Program (Appendix)

09676175 .092900

7. ______ Nucleotide and/or Amino Acid Sequence Submission
(if applicable, all necessary)
a. ____ Computer Readable Copy
b. ____ Paper Copy (identical to computer copy)
c. ____ Statement verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

8. ______ Assignment Papers (cover sheet & documents(s))
9. ______ a. 37 CFR 3.73(b) Statement (where there is an assignee)
______ b. Power of Attorney
10. ______ English Translation Document (if applicable)
11. ______ a. Information Disclosure Statement (IDS)/PTO-1449
______ b. Copies of IDS Citations
12. ______ Preliminary Amendment
13. ______ Return Receipt Postcard (MPEP 503) (Should be specifically itemized)
14. ______ a. Small Entity Statement(s)
b. Statement filed in prior application, Status still proper and desired
15. ______ Certified Copy of Priority Document(s) (if foreign priority is claimed)

16. X Other: Express Mail Certificate with copy of postcard showing contents of Express Mail package.

17. **If a CONTINUING APPLICATION,** check appropriate box and supply the requisite information:
____ Continuation ____ Divisional ____ Continuation-in-part (CIP)
of prior application No: _

18. **Correspondence Address**
_____ Customer Number or Bar Code Label ______________________
(Insert Customer No. or Attach Bar Code Label here)
or
X Correspondence Address Below

[illegible] 9/29/00

NAME John P. Ward Reg. No. 40,216
BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
ADDRESS 12400 Wilshire Boulevard
Seventh Floor
CITY Los Angeles STATE California ZIP CODE 90025-1026
Country U.S.A. TELEPHONE (408) 720-8598 FAX (408) 720-9397

FEE TRANSMITTAL FOR FY 2000

TOTAL AMOUNT OF PAYMENT ($) 690.00

Complete if Known:

Application No.	Not Yet Assigned
Filing Date	Not Yet Assigned
First Named Inventor	Orna Etzion
Group Art Unit	Not Yet Assigned
Examiner Name	Not Yet Assigned
Attorney Docket No.	042390.P7512

METHOD OF PAYMENT (check one)

1. [x] **The Commissioner is hereby authorized to charge indicated fees and credit any over payments to:**

Deposit Account Number 02-2666
Deposit Account Name ______

[x] **Charge Any Additional Fee Required Under 37 CFR 1.16 and 1.17**

2. x **Payment Enclosed:**
x **Check**
____ **Money Order**
____ **Other**

FEE CALCULATION

1. BASIC FILING FEE

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
101	690	201	345	Utility application filing fee	690.00
106	310	206	155	Design application filing fee	
107	480	207	240	Plant filing fee	
108	690	208	345	Reissue filing fee	
114	150	214	75	Provisional application filing fee	

SUBTOTAL (1) $ 690.00

2. EXTRA CLAIM FEES

			Extra Claims		Fee from below		Fee Paid
Total Claims	15	−20** =	0	X	18.00	=	0
Independent Claims	3	−3** =	0	X	78.00	=	0
Multiple Dependent						=	

****Or number previously paid, if greater; For Reissues, see below.**

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description
103	18	203	9	Claims in excess of 20
102	78	202	39	Independent claims in excess of 3
104	260	204	130	Multiple dependent claim, if not paid
109	78	209	39	**Reissue independent claims over original patent
110	18	210	9	**Reissue claims in excess of 20 and over original patent

SUBTOTAL (2) $ 0

Patent fees are subject to annual revisions. Small Entity payments must be supported by a small entity statement, otherwise large entity fees must be paid.
See Forms PTO/SB/09-12

09/482659 ·012500

FEE CALCULATION (continued)

3. ADDITIONAL FEES

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
105	130	205	65	Surcharge - late filing fee or oath	
127	50	227	25	Surcharge - late provisional filing fee or cover sheet	
139	130	139	130	Non-English specification	
147	2,520	147	2,520	For filing a request for reexamination	
112	920*	112	920*	Requesting publication of SIR prior to Examiner action	
113	1,840*	113	1,840*	Requesting publication of SIR after Examiner action	
115	110	215	55	Extension for response within first month	
116	380	216	190	Extension for response within second month	
117	870	217	435	Extension for response within third month	
118	1,360	218	680	Extension for response within fourth month	
128	1,850	228	925	Extension for response within fifth month	
119	300	219	150	Notice of Appeal	
120	300	220	150	Filing a brief in support of an appeal	
121	260	221	130	Request for oral hearing	
138	1,510	138	1,510	Petition to institute a public use proceeding	
140	110	240	55	Petition to revive unavoidably abandoned application	
141	1,210	241	605	Petition to revive unintentionally abandoned application	
142	1,210	242	605	Utility issue fee (or reissue)	
143	430	243	215	Design issue fee	
144	580	244	290	Plant issue fee	
122	130	122	130	Petitions to the Commissioner	
123	50	123	50	Petitions related to provisional applications	
126	240	126	240	Submission of Information Disclosure Stmt	
581	40	581	40	Recording each patent assignment per property (times number of properties)	
146	690	246	345	For filing a submission after final rejection (see 37 CFR 1.129(a))	
149	690	249	345	For each additional invention to be examined (see 37 CFR 1.129(a))	

Other fee (specify) ____________________ ______

Other fee (specify) ____________________ ______

SUBTOTAL (3) $ 0

*Reduced by Basic Filing Fee Paid

SUBMITTED BY:

Typed or Printed Name: John P. Ward

Signature [illegible] **Date** 9/29/00

Reg. Number 40,216 **Deposit Account User ID** 02-2666 (complete if applicable)

42390.P7512 Patent

APPLICATION FOR UNITED STATES LETTERS PATENT

FOR

A Method and Apparatus for Generating An Expected Top Of Stack During Instruction Translation

Inventor
Orna Etzion

Prepared by:
Blakely, Sokoloff, Taylor & Zafman
1279 Oakmead Parkway
Sunnyvale, California 94086
(408) 720-8598

"Express Mail" mailing label number EL 617 211 053 US
Date of Deposit September 29, 2000
I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Assistant Commissioner of Patents and Trademarks, Washington, D.C. 20231.

Lisa Kaiser
(Typed or printed name of person mailing paper or fee)

Lisa Kaiser
(Signature of person mailing paper or fee)

A Method and Apparatus for Generating An Expected Top Of Stack During Instruction Translation

FIELD OF THE INVENTION

The present invention relates to the field of computers. In particular the present invention discloses a method and apparatus for generating an expected Top of Stack during instruction translation.

BACKGROUND OF THE INVENTION

New computer architecture designs continue to improve the processing performance of computer processors. However, the existing computer programs often cannot be directly executed on new computer architectures.

To execute computer programs written for older computer processor architectures on a newer computer processors architectures, computer systems may perform binary translation. A binary translator translates blocks of code written for older processor architecture into equivalent blocks of code that can be executed on a newer processor architecture. The translation can be done "on the fly" (i.e., while executing the older architecture code). Translation could also be done before execution.

Binary translators typically insert extra instructions as necessary to ensure the instructions written for the older processor architecture will function properly on the newer processor architecture. Since the extra instructions added are not directly related to the goal of the original program, these extra instructions are "overhead" code that reduces performance. It would be desirable to implement translators in a manner that reduces the amount of overhead code.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a flow diagram describing the steps of generating an expected top of stack (TOS) during binary translation according to one embodiment.

Figure 2 is a block diagram illustrating binary translated pseudo code that includes an expected TOS in accordance with one embodiment.

Figure 3 is a block diagram illustrating binary translated pseudo code that includes an expected TOS in accordance with one embodiment.

Figure 4 illustrates a computer system having a computer readable medium with instructions stored thereon according to one embodiment.

DETAILED DESCRIPTION

A method and apparatus for generating an expected top of stack (TOS) during translations is disclosed. In the following description, for purposes of explanation, specific nomenclature is set forth to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that these specific details are not required in order to practice the present invention.

As previously set forth, legacy applications that were written for an older processor architecture can be run on a new processor architecture by having instructions translated. During translation, "overhead" code is typically added to have the applications executable on the new processor architecture. The present invention provides a technique for reducing the overhead.

In particular, programs commonly use a technique referred to as stacks when operating on multiple units of data. Stacks are typically Last In First Out (LIFO) based (i.e., the most recent data operand placed (pushed) onto the stack is the first item removed (popped) from the stack). Moreover, the most recent data operand pushed onto the stack is also referred to as the top of the stack (TOS). Stacks can be implemented in reserved area of memory or may be implemented using a set of registers.

By way of example, a stack may include five entries (ST0 – ST4). Initially, the stack is empty. An instruction may perform an operation and push the result onto the stack at ST0. A second instruction may push a second result onto the stack at ST1. A third instruction may perform a third operation that involves using the most recent entry of the stack (ST1) (i.e., the TOS). As a result, the third instruction pops the result stored at ST1. Therefore, following the third instruction, the TOS is ST0.

Moreover, instructions for older processor architecture that refer to the stacks commonly use instructions that are TOS based. They do not refer to specific stack positions, but refer to TOS, TOS-1 etc. In such an embodiment, the processor includes logic to maintain/keep track of the current TOS position (i.e., knowing which of the physical entries (e.g., ST0-ST4) is currently the TOS).

As such, the TOS is maintained as execution of the program proceeds from one block of instructions (BB1) to a second block of instructions (BB2). A block of instructions typically consists of a sequence of contiguous instructions that ends in a branch instruction, which if taken passes execution to a second block instructions that typically does not immediately follow in the sequence of instructions in the program.

For example, an "in-order" sequence of a program may consist of BB1, BB2, BB3, BB4 . . . If the branch instruction of BB1 is taken, the flow of execution may be specified to jump to BB3 (or another BB that does not immediately follow BB1 in the sequence of BBs in the application program). If the branch instruction of BB1 is not taken, then execution would presumably continue at BB2. Moreover, when jumping from BB1 to BB3 (for example), the TOS continues to be maintained by the processor (in the case of the older processor architecture), so that BB3 may use the stack, by simply making reference to the TOS.

Some new processor architectures, however, may not provide automatic tracking of the TOS (as described above for the older processor architecture). As a result, rather than simply pushing and poping operands on/off a stack by referring to the TOS (without having to identify the actual physical entry/position of the TOS), the instructions make specific reference to the physical location of the TOS. More specifically, consider the example of a set of registers being set aside for the stack, but no logic of the processor monitors which register contains the current TOS. Therefore, the instructions for the new processor architecture specify the register from which data is to be removed (i.e., popped), rather than simply requesting the data at the TOS be popped, as in the case of the instructions executable on the older processor architecture.

As a result, when translating instructions from being executable on the older processor architecture (which made reference to the TOS), the translated instructions make specific reference to the register/memory location that is assumed to be the TOS. Moreover, the translated BBs, previously, would make the assumption that the TOS remained the same (i.e., the net result of a preceding BB's did not increase or decrease the position of the TOS). However, the assumption that the TOS has not changed is relatively frequently incorrect. As a result, before a translated BB could be executed, the stack would need to be rotated to have the actual TOS correspond to the expected TOS (which results in overhead). The stack would need to be rotated because the translated instructions have already been translated to make reference to specific locations that were believed to be the TOS.

In one embodiment of the present invention, however, during the translation of a program, an expected TOS for each respective BB is generated. By generating, during translation, an expected TOS for a BB, less overhead is incorporated into the actual execution of the translated BB. In particular, at the time of executing the translated BB, there is a greater likelihood that the expected TOS will match the actual TOS, thereby avoiding the need to rotate the stack to have the expected TOS match the actual TOS.

Generating Expected TOS During Translation

Figure 1 is a flow diagram describing the steps of generating an expected TOS during translation according to one embodiment. In block 102, a block of instructions (BB) is received/identified to be translated from being executable on a first processor architecture into instructions executable on a second processor architecture.

In block 104 of the flow diagram, the expected TOS for the respective BB, is determined. In one embodiment, a dynamic translator uses the first run-time entry state, which is already known when the respective block is about to be translated. In case of a static translator, the expected TOS, in one embodiment, is determined by a instruction analyzer counting the number of times data is pushed onto the stack, and the number of times data is removed/popped from the stack.

In block 106, the BB identified/received to be translated is translated from being executable on a first processor architecture into instructions executable on a second processor architecture. The translation may be done dynamic or via a static binary translator. Moreover, in alternative embodiments, architectural simulators or virtual-machine implementations using similar, code-generation-based techniques, may also be used to transform the basic blocks of instructions, without departing from the invention.

In block 108, it is determined whether the net result of the respective BB changes the TOS. If the net result of the respective BB is to change the TOS (by increasing or decreasing the position of the TOS), in block 110 at least one instruction is added to the end of the BB to update the actual TOS accordingly. In one embodiment, the actual TOS can be stored in a register. On the other hand, if the net effect of the respective BB is to not change the TOS, then the additional instruction to update the actual TOS is not added.

In block 112, an instruction is added to the beginning of the present BB being translated, to compare the expected TOS with the Actual TOS. If the expected TOS is not equal to the Actual TOS, a TOS correction handler is called to correct the actual TOS to be aligned with the expected TOS. In one embodiment, the correction handler includes generating the delta of the expected TOS and the actual TOS. The stack is then adjusted/rotated by the delta.

As a result, the penalty of the difference between the actual TOS and the expected TOS does not propagate to subsequent BBs. When execution proceeds to the next BB to be executed, the correction of the actual TOS is already done, and the stack-depth expected by the next BB should match the actual depth. In the present invention, by generating an expected TOS during translation, the likelihood of the expected TOS and actual TOS not matching is reduced. As a result, the need to rotate the actual stack during execution is reduced.

Thereafter, in block 114, if additional BBs remain to be translated, the translator continues to repeat blocks 102-112 for the remaining BBs that are to be translated.

Figure 2 illustrates a pseudo code example of one embodiment where the expected TOS is equal to the Actual TOS. In block 202, the entry conditions for BB1 are presented. In the entry conditions, the expected TOS for BB1 is 5, as is the actual TOS. The translated pseudo-code of BB1, shown in block 204, includes the compare instruction to determine if the expected TOS is equal to the Actual TOS. If the expected TOS is not equal to the actual TOS, the correction handler is called to correct the actual TOS by rotating the stack accordingly.

On the other hand, if the Actual and expected TOS are equal, the instructions of BB1 are executed. Afterwards, the updated TOS as generated by the execution of the instructions in BB1 is assigned as the Actual TOS. In the case BB1, of the net effect of the code is to push one additional data entry onto the stack. Thus, following execution of BB1, as shown in block 206, the expected TOS is 6 and the Actual TOS is 6.

Execution of the program continues at the translated BB2 as shown in block 208. Execution of BB2 generates the conditions as shown in block 210. Moreover, in block 208, the net use of the stack does not change the TOS. Thus, the actual TOS is not changed (i.e., BB2 pops an entry from the stack and pushes an entry onto the stack).

Figure 3 further illustrates an additional pseudo code example of one embodiment where the expected TOS does not equal to the Actual TOS. In block 302, the entry condition of BB1 is presented. In the entry condition, the expected TOS is 5, and the actual TOS is 4. In block 304, the first line of the translated pseudo code of BB1 compares the expected TOS with the actual TOS. Because the they are not equal, the correction pseudo code in block 306 is called to correct the actual TOS as shown in block 308. Thereafter, execution of BB1 proceeds as shown in block 310.

Once again, the pseudo code of BB1 generates a net of adding one entry to the stack. Thus, the actual TOS is updated to 6, at the end of BB1. In block 314, the result of the pseudo code of BB2 does not change the TOS. Thus, the actual TOS is not changed, as shown in block 316.

The foregoing has described a method and apparatus for generating an expected TOS during binary translation. It is contemplated that changes and modifications may be made by one of ordinary skill in the art, to the materials, arrangements, and code listings of the present invention without departing from the scope of the invention. For example, the methods as described above, including the methods of generating an expected TOS, in one embodiment, could be performed when compiling source code in prior to executing the source code.

In addition, the methods as described above, can be stored in memory of a computer system as a set of instructions to be executed. In addition, the instructions to perform the methods as described above could alternatively be stored on other forms of computer-readable medium, including magnetic and optical disks. For example, method of the present invention can be stored on computer-readable mediums, such as magnetic disks or optical disks, that are accessible via a disk drive (or computer-readable medium drive), such as the disk drive shown in Figure 4.

Alternatively, the logic to perform the methods as discussed above, including the methods of generating an expected TOS during binary translation, could be implemented in discrete hardware components such as large-scale integrated circuits (LSI's), application-specific integrated circuits (ASIC's) or in firmware such as electrically erasable programmable read-only memory (EEPROM's).

CLAIMS

We claim:

1. A method of translating instructions, said method comprising:

translating a first block of instructions executable in a first processor architecture, into a translated first block of instructions executable in a second processor architecture, said translated first block of instructions operating with a stack of data entry positions; and

generating an expected Top of Stack (TOS) position in said stack for said first block of code.

2. The method as claimed in claim 1, said method further comprising:

adding at least one instruction to said translated first block of instructions to determine if said first expected TOS is equal to an actual TOS at a time of executing said translated first block of instructions.

3. The method as claimed in claim 2, wherein said instruction added to said first block of instructions, branches to correction code if said expected TOS is not equal to said actual TOS.

4. The method as claimed in claim 3, said method further comprising:

determining if execution of instructions in said first block of instructions changes the actual TOS.

5. The method as claimed in claim 4, said method further comprising:

in response to determining execution of instructions in said first block of instructions changes the actual TOS, adding an instruction to an end of the first block of instructions to update the actual to TOS.

6. A computer-readable medium having stored thereon a set of instructions to translate instructions, said set of instructions, which when executed by a processor, cause said processor to perform a method comprising:

translating a first block of instructions executable in a first processor architecture, into a translated first block of instructions executable in a second processor architecture, said translated first block of instructions operating with a stack of data entry positions; and

generating an expected Top of Stack (TOS) position in said stack for said first block of code.

7. The computer-readable medium as claimed in claim 6, wherein said set of instructions further includes additional instructions, which when executed by said processor, cause said processor to perform said method further comprising:

adding at least one instruction to said translated first block of instructions to determine if said first expected TOS is equal to an actual TOS at a time of executing said translated first block of instructions.

8. The computer-readable medium as claimed in claim 7, wherein said instruction added to said first block of instructions, branches to correction code if said expected TOS is not equal to said actual TOS.

9. The computer-readable medium as claimed in claim 8, wherein said set of instructions further includes additional instructions, which when executed by said processor, cause said processor to perform said method further comprising:

determining if execution of instructions in said first block of instructions changes the actual TOS.

10. The computer-readable medium as claimed in claim 9, wherein said set of instructions further includes additional instructions, which

when executed by said processor, cause said processor to perform said method further comprising:

in response to determining execution of instructions in said first block of instructions changes the actual TOS, adding an instruction to an end of the first block of instructions to update the actual to TOS.

11. A system comprising:

a first unit of logic to translate a first block of instructions executable in a first processor architecture, into a translated first block of instructions executable in a second processor architecture, said translated first block of instructions operating with a stack of data entry positions; and

a second unit of logic to generate an expected Top of Stack (TOS) position in said stack for said first block of code.

12. The system as claimed in claim 11, wherein said second unit of logic further adds at least one instruction to said translated first block of instructions to determine if said first expected TOS is equal to an actual TOS at a time of executing said translated first block of instructions.

13. The system as claimed in claim 12, wherein said instruction added to said first block of instructions, branches to correction code if said expected TOS is not equal to said actual TOS.

14. The system as claimed in claim 13, wherein said second unit of logic determines if execution of instructions in said first block of instructions changes the actual TOS.

15. The system as claimed in claim 14, wherein said second unit of logic, in response to determining execution of instructions in said first block of instructions changes the actual TOS, adds an instruction to an end of the first block of instructions to update the actual to TOS.

ABSTRACT OF THE DISCLOSURE

A method of generating an expected TOS during translation of instructions. The method includes translating a first block of instructions executable in a first processor architecture, into a translated first block of instructions executable in a second processor architecture, wherein the translated first block of instructions operate with a stack of data entry positions. During the translation, an expected Top of Stack (TOS) position in the stack for the first block of code is generated.

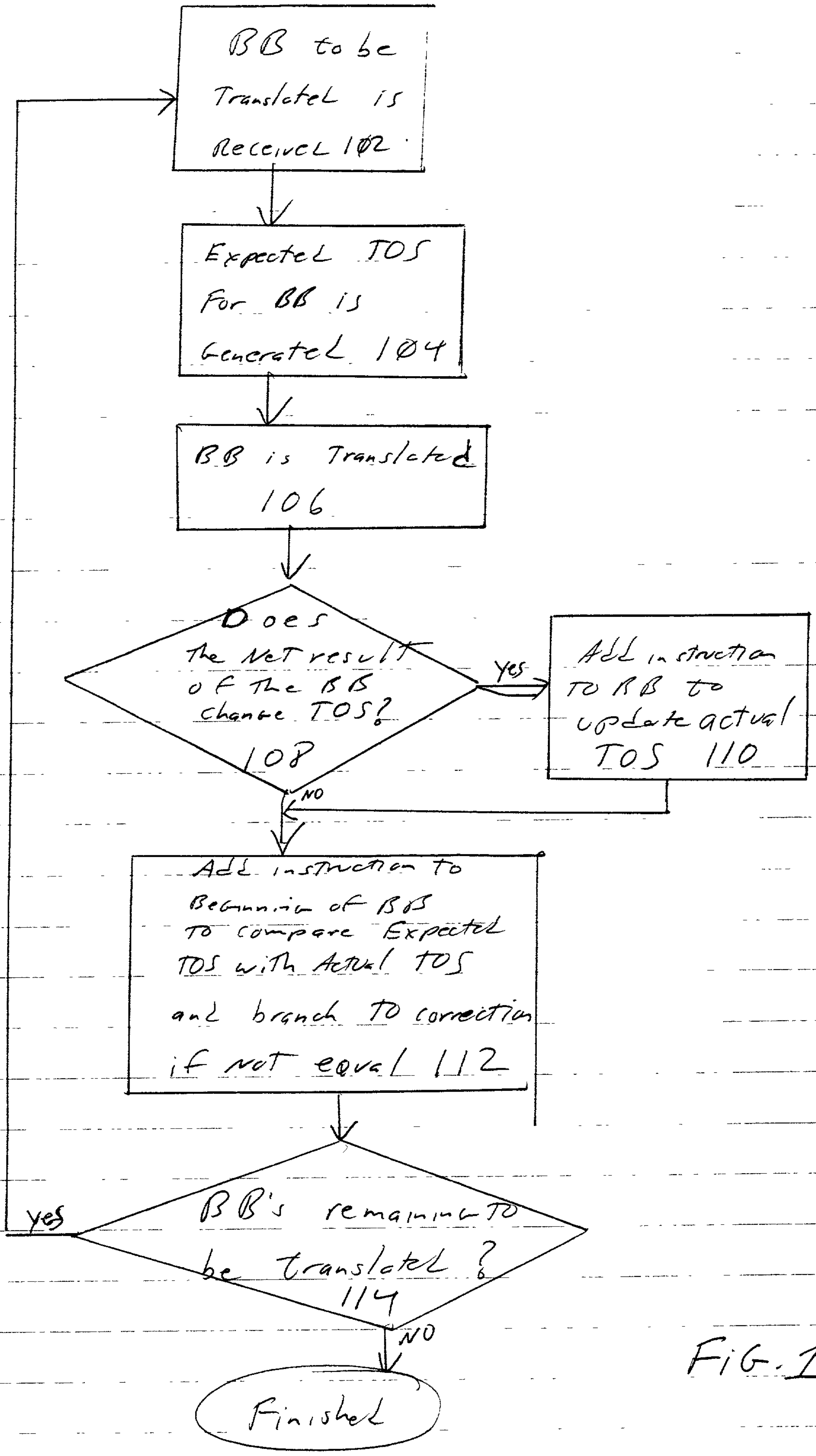
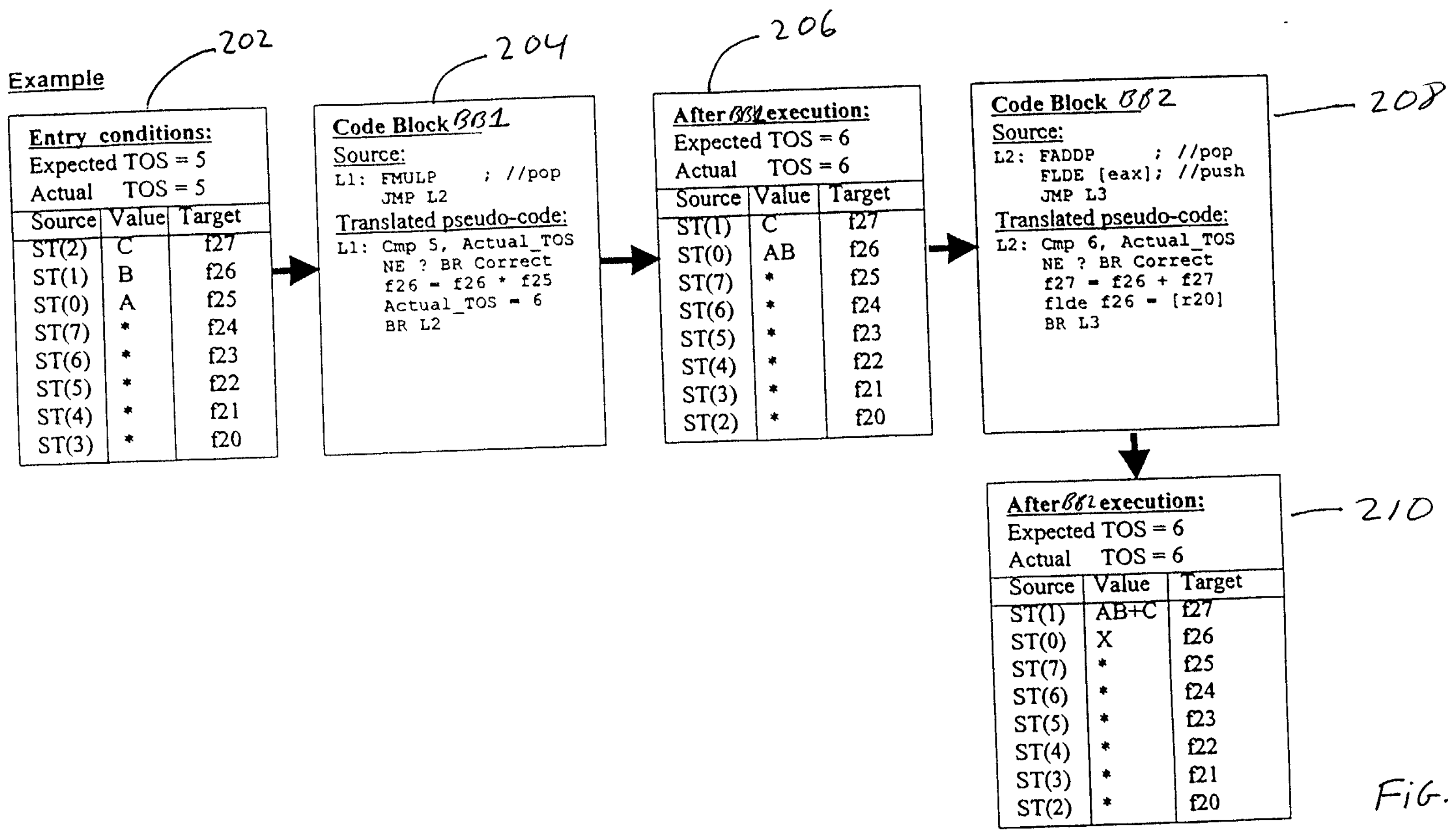

BB to be Translated is Received 102
Expected TOS For BB is Generated 104
BB is Translated 106
Does The Net result of The BB change TOS? 108
yes
Add instruction To BB to update actual TOS 110
NO
Add instruction to Beginning of BB To compare Expected TOS with Actual TOS and branch TO correction if not equal 112
BB's remaining To be translated? 114
yes
NO
Finished

FIG. 1



Entry conditions:
Expected TOS = 5
Actual TOS = 5

Source	Value	Target
ST(2)	C	f27
ST(1)	B	f26
ST(0)	A	f25
ST(7)	*	f24
ST(6)	*	f23
ST(5)	*	f22
ST(4)	*	f21
ST(3)	*	f20

Code Block BB1
Source:
```
L1: FMULP     ; //pop
    JMP L2
```
Translated pseudo-code:
```
L1: Cmp 5, Actual_TOS
    NE ? BR Correct
    f26 = f26 * f25
    Actual_TOS = 6
    BR L2
```

After BB1 execution:
Expected TOS = 6
Actual TOS = 6

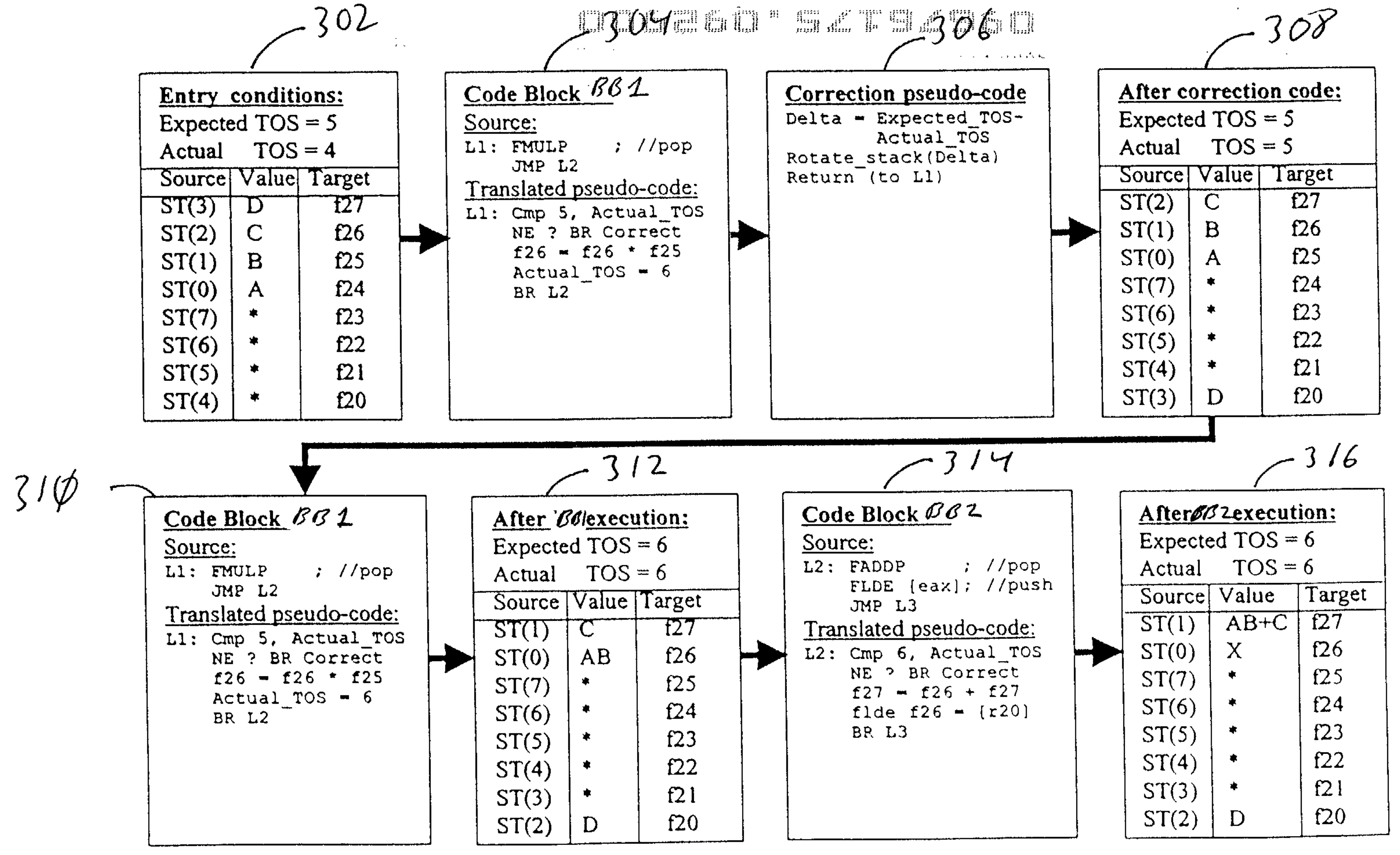
Source	Value	Target
ST(1)	C	f27
ST(0)	AB	f26
ST(7)	*	f25
ST(6)	*	f24
ST(5)	*	f23
ST(4)	*	f22
ST(3)	*	f21
ST(2)	*	f20

Code Block BB2
Source:
```
L2: FADDP      ; //pop
    FLDE [eax]; //push
    JMP L3
```
Translated pseudo-code:
```
L2: Cmp 6, Actual_TOS
    NE ? BR Correct
    f27 = f26 + f27
    flde f26 = [r20]
    BR L3
```

After BB2 execution:
Expected TOS = 6
Actual TOS = 6

Source	Value	Target
ST(1)	AB+C	f27
ST(0)	X	f26
ST(7)	*	f25
ST(6)	*	f24
ST(5)	*	f23
ST(4)	*	f22
ST(3)	*	f21
ST(2)	*	f20

FIG. 2



Entry conditions:
Expected TOS = 5
Actual TOS = 4

Source	Value	Target
ST(3)	D	f27
ST(2)	C	f26
ST(1)	B	f25
ST(0)	A	f24
ST(7)	*	f23
ST(6)	*	f22
ST(5)	*	f21
ST(4)	*	f20

Code Block BB1
Source:
```
L1: FMULP     ; //pop
    JMP L2
```
Translated pseudo-code:
```
L1: Cmp 5, Actual_TOS
    NE ? BR Correct
    f26 = f26 * f25
    Actual_TOS = 6
    BR L2
```

Correction pseudo-code
```
Delta = Expected_TOS-
          Actual_TOS
Rotate_stack(Delta)
Return (to L1)
```

After correction code:
Expected TOS = 5
Actual TOS = 5

Source	Value	Target
ST(2)	C	f27
ST(1)	B	f26
ST(0)	A	f25
ST(7)	*	f24
ST(6)	*	f23
ST(5)	*	f22
ST(4)	*	f21
ST(3)	D	f20

Code Block BB1
Source:
```
L1: FMULP     ; //pop
    JMP L2
```
Translated pseudo-code:
```
L1: Cmp 5, Actual_TOS
    NE ? BR Correct
    f26 = f26 * f25
    Actual_TOS = 6
    BR L2
```

After BB1 execution:
Expected TOS = 6
Actual TOS = 6

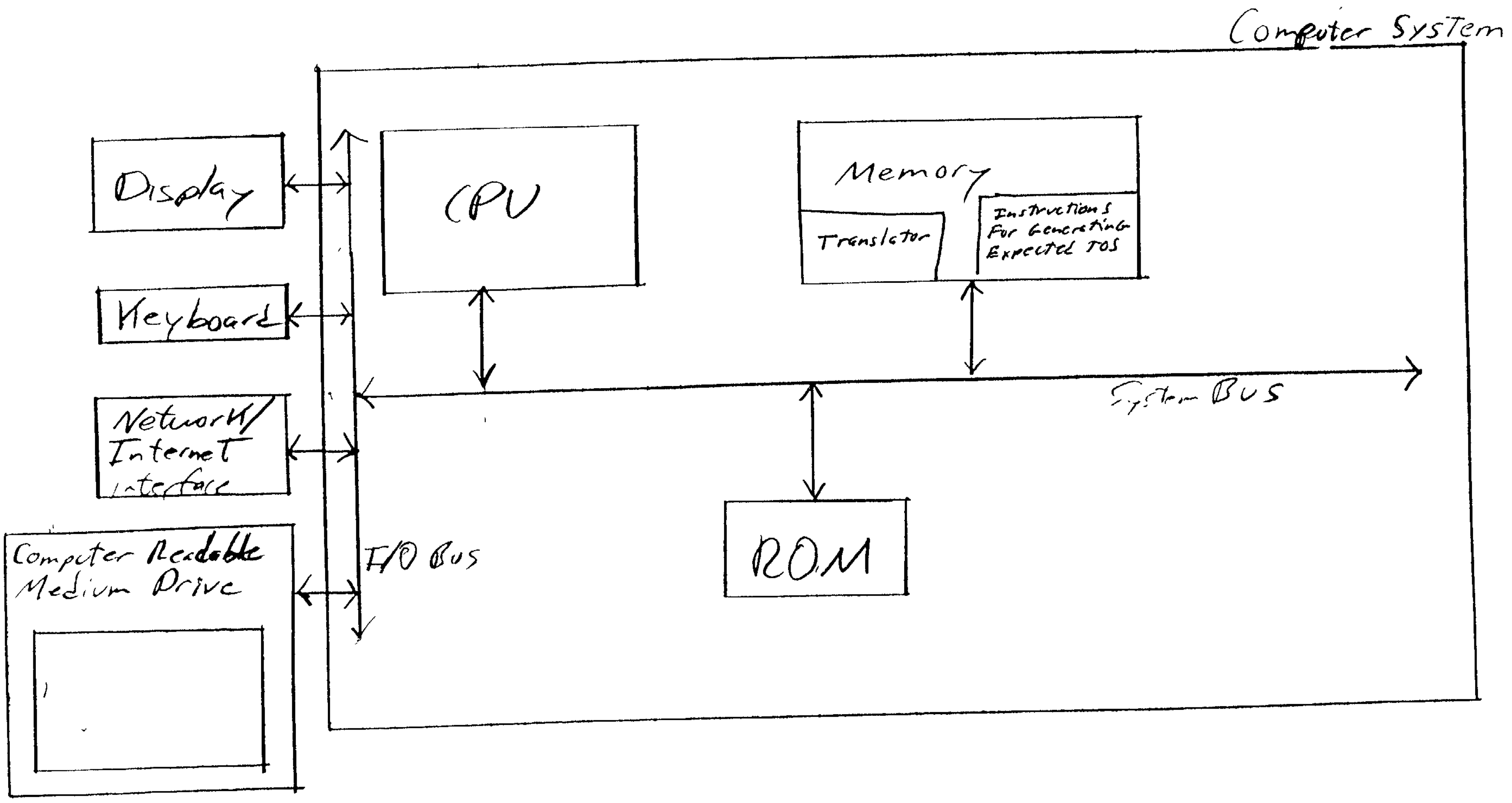
Source	Value	Target
ST(1)	C	f27
ST(0)	AB	f26
ST(7)	*	f25
ST(6)	*	f24
ST(5)	*	f23
ST(4)	*	f22
ST(3)	*	f21
ST(2)	D	f20

Code Block BB2
Source:
```
L2: FADDP       ; //pop
    FLDE [eax]; //push
    JMP L3
```
Translated pseudo-code:
```
L2: Cmp 6, Actual_TOS
    NE ? BR Correct
    f27 = f26 + f27
    flde f26 = [r20]
    BR L3
```

After BB2 execution:
Expected TOS = 6
Actual TOS = 6

Source	Value	Target
ST(1)	AB+C	f27
ST(0)	X	f26
ST(7)	*	f25
ST(6)	*	f24
ST(5)	*	f23
ST(4)	*	f22
ST(3)	*	f21
ST(2)	D	f20

FiG. 3



Computer System
Display
Keyboard
Network/ Internet interface
Computer Readable Medium Drive
I/O Bus
CPU
Memory
Translator
Instructions For Generating Expected TOS
System Bus
ROM

Fig. 4